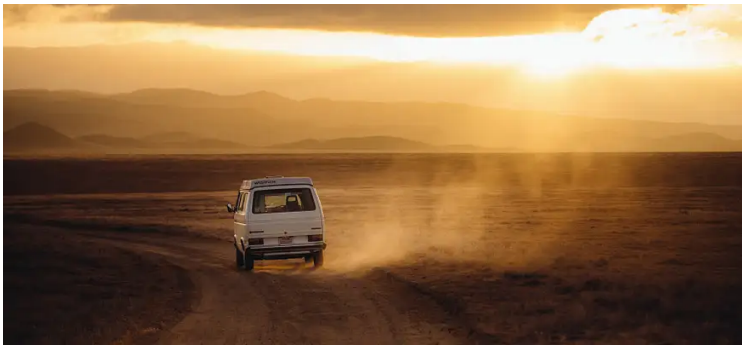
Making Custom REST Routes In WordPress 4.4 and Beyond



DAVID HAYES / UPDATED: JUNE 21, 2017
Posted In: Back-End Development
Difficulty: Advanced

In WordPress 4.4, some significant parts of the long awaited REST API made it into WordPress core. While for many use cases you'll probably need to keep using the plugin until at least WordPress 4.7 comes out, you can do some cool things with the "infrastructure" for the API that did make it into core, in particular the register_rest_route function.

MAIN TOPICS:

- *Back-End Development*
- *Courses*
- *Editorial*
- *Front-End Development*
- *Quick Guides*
- *Server Administration*
- *Unfiled*
- *Using WordPress*

POPULAR POSTS

- *Course: Learn WordPress Development*
- *Complete Guide to WordPress Security*
- *WordPress Page Builders, Reviewed*
- *Best Cloud Hosting*
- *Siteground Review*
- *Best WordPress Hosting*
- *How to Start A*

Derisively, someone quipped at the height of frustration around the 4.5 merge discussion (perhaps best captured by Brian Krogsgard's post about it) that if the only thing that the REST API managed to land in WordPress core was to make a better way to create AJAX endpoints, that'd be a real disappointment. It's hard to disagree with the sentiment.

But that said, it's pretty cool to get to use `register_rest_route` today. It's not the only feature that would be cool to get from the REST API, but it is the one we've got today and it does enable some pretty cool stuff. Let's get to it!

A Conceptual Summary of a "REST Route"

I wrote a few weeks ago about the concepts behind REST and what we mean by a route. Please read that article if you're not clear about the concepts, because here we'll be kind of brief. The executive summary of that article is this:

- REST is a URL and resource-oriented way of presenting access

n API

rces are things like posts, pages, or tags

s are the URLs at which you manipulate resources

- The same URL for a REST API will respond differently to different HTTP methods

If that all make sense enough to you now, please keep reading. If not, do make sure you get clear on that. It's so useful to be able to answer your friend asking you, "But Seriously, What is a REST API Anyway?"

Basics of Registering a Route with register_rest_route

If you're pretty clear about all the REST ideas, most of `register_rest_route` makes pretty good sense. If you're not, it can be mystifying. The function itself will look something like this in use:

```
add_action( 'rest_api_init',
'wpshout_register_routes' );
```

```
function
wpshout_register_routes() {
    register_rest_route(
        'myplugin/v1',
        '/author/(?P<id>\d+)',
        array(
            'methods' => 'GET',
            'callback' =>
'wpshout_find_author_post_title
',
        )
    );
}
```

As seasoned WordPress developers will recognize, we're hooking on to an action with a function. In our case that action is `rest_api_init`. As you probably guessed that joined WordPress in 4.4 with the `register_rest_route` function itself. Then we're calling the actual function from inside that action, and we're passing three arguments:

1. The "namespace" to our endpoint. This is the "preamble" to the actual name of our resource. Versioning via URL (thus including something like `/v1`) is common but not required.

2. The actual "route" to our endpoint. This is a little scary looking. The reason is that it's using regular expressions *with a named capture group*. Two kind-of-scary things. An exhaustive summary of that will have to wait to another time. Just know that `author/(?P<id>\d+)` is going to match things like `author/123` and it'll "capture" the `123` as the value for `id`. Make sense?
3. This is the typical WordPress `$args` array. The two methods our sample code specifies are common and basically required: the HTTP methods this specific "route" should handle, and the function by which it should handle them.

Ok. Now, what about the actual function that responds to a `GET` request to `http://wwww.example.com/wp-json/myplugin/v1/author/123`? Well it'll look something like this:

```
function wpshout_find_author_post_title( $data ) {
```

```
    $posts = get_posts( array(
        'author' =>
$data['id'],
    ) );

    if ( empty( $posts ) ) {
        return null;
    }

    return $posts[0]-
>post_title;
}
```

The first thing to notice about our function is that it receives a `$data` array, which contains our named capture group — `id`. This is why it's useful: we can then use that captured item to do cool stuff. Like, in this example, get posts by an author based on their WordPress user ID. This function's pretty simple, but it's for example purposes only. This theoretical endpoint just responded with the title of the most recent blog post that the user has authored. It's useful, but it's just the start of the cool stuff you can do with REST endpoints.

Another thing to realize is that these are endpoint functions where you

return data that WordPress's infrastructure will take care of serializing out to JSON for you. This is one of the significant ways that this is a departure (and for me a big improvement on the classic "admin ajax"): you don't need a `die` and you don't echo your result. Instead you return it as a raw PHP structure and WordPress converts it to JSON for you in the background. So, with this example code, because it's returning a string, we'll just see a quoted string on the endpoint — it will not be nested inside of a JSON object or anything unless you put it in one.

Going a Step Further

This isn't going to a be an exhaustive tour of the realities of `register_rest_route`. While I've toyed with it a fair amount, I've not yet had the opportunity to use it in a project that it made sense for. (If you're looking for help with one you know it does, Press Up has some availability in Q3 2016 and beyond. 😉 Get in touch!)

That said, there are some clear quick things that I do want to mention about

using `register_rest_route` effectively.

OTHER OR MULTIPLE RESPONSE METHODS

The first thing to keep in mind is that REST is all about HTTP methods and how you respond to them with different action inside the system. What this means is that you may, as we discussed in the REST overview article, want to accept `PUT`, `PATCH`, and `POST` requests for your endpoints. There are two ways you can do that. The first is pretty clear and explicit:

```
register_rest_route(
'myplugin/v1', '/update/(?
P<id>\d+)', array(
    'methods' => 'POST, PUT,
PATCH',
    'callback' =>
'wpshout_special_update_functio
n',
) );
```

But alternatively, you can use a helper constant from the `WP_REST_Server` class, like so:

```
register_rest_route(
'myplugin/v1', '/update/(?
P<id>\d+)', array(
    'methods' =>
WP_REST_Server::EDITABLE,
    'callback' =>
'wpshout_special_update_functio
n',
) );
```

These two are functionally equivalent, but the latter is a little simpler and clearer to my eyes. Especially for people who don't speak REST or HTTP fluently. There are five such helpful constants on the class:

- `READABLE = 'GET'`
- `CREATABLE = 'POST'`
- `EDITABLE = 'POST, PUT, PATCH'`
- `DELETABLE = 'DELETE'`
- `ALLMETHODS = 'GET, POST, PUT, PATCH, DELETE'`

At the end of the day, it's your choice if you use them or not. But it's good to know what that `WP_REST_Server::ALLMETHODS` means whether or not you use them yourself.

THE CONTROLLER PATTERN

The last thing to understand is the beneficial role that the containment inside a PHP object can have for something like a set of REST routes that all map to a single entity in your system. If you make a series of functions to create, read, update, and delete your resources, it's nice to have those structurally close to each other.

While our examples are just simple functions as most WordPress developers have done for years, keeping them together when you have more than two in an object is hugely beneficial for you and other people who may later read your code.

Inside of the plugin-side of the REST API, there exists a class `WP_REST_Controller` which you can (and perhaps should) extend when working on a site where you can count on it. But if not, you can pretty quickly and easily make your own routes and use them effectively inside an object. You just have to get used to defining your callback functions in object terms, so for an old-school function/procedure you'd use:

```
    'callback' =>
'wpshout_special_update_functio
n',
```

Inside an object it'd look more like:

```
    'callback' => array( $this,
'update_item' )
```

There's a great deal more that's possible/interesting/easy about using a controller if you do extend (the plugin's) `WP_REST_Controller` class, for which you should head over the plugin's documentation. That page, in particular, is a very rich thought of thinking about using `wp_rest_route`.

Now, Go Forth and Admin-Ajax No More!

I personally have a not-so-fond relationship with WordPress's `admin-ajax` system. I think it's served us reasonably well for the last decade or so, but I'm happy to be able to today know that I can reach into WordPress (without a plugin) and create a way to

quickly serve data to the world that doesn't use it.

The final REST API, when it lands in WordPress, will likely have some pretty far-reaching effects. But `register_rest_route`, while it's not the 800 pound gorilla everyone has been waiting for, is a pretty great little feature in its own right. Try it out, and happy hacking!

Image credit: Forrest Cavale via Unsplash

Get the very latest WordPress development tutorials, tricks and news in your inbox.

Enter your email address here...

Sign Up Now

Resources

- ARTICLES
- VIDEO QUICK GUIDES
- BEST WORDPRESS HOSTING
- RECOMMENDED PRODUCTS

Courses

- WORDPRESS SECURITY WITH CONFIDENCE
- UP AND RUNNING WP
- FREE WPSHOUT COURSES
- WORDPRESS HOSTING GUIDE

About Us

- ABOUT & CONTACT
- SPONSOR WPSHOUT

Most Recent

- PHP FOR BEGINNERS: STARTING ON BACKEND WORDPRESS DEVELOPMENT
- WORDPRESS CUSTOM TAXONOMIES: HOW AND WHY TO CREATE THEM
- LEARN WORDPRESS DEVELOPMENT: THE BASIC COURSE